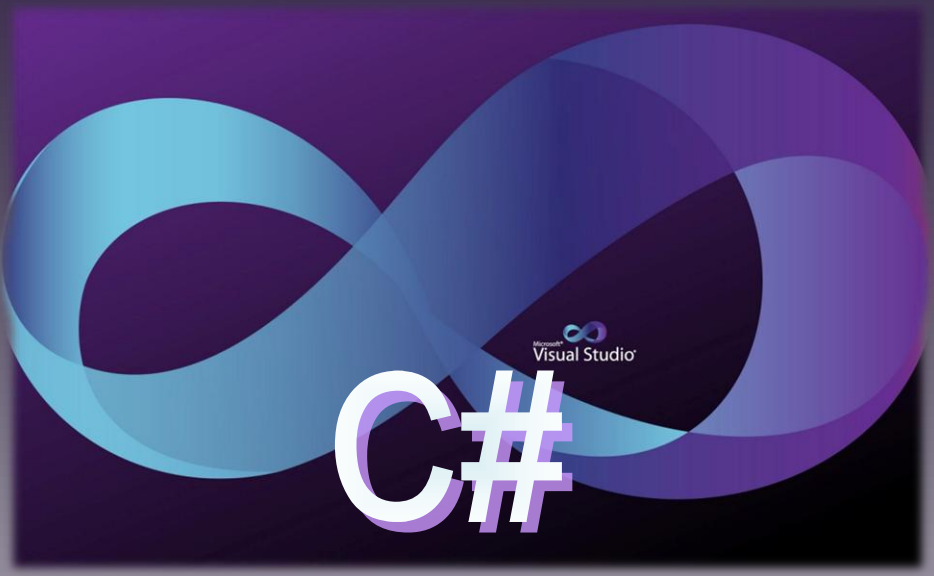
ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI



CORSO DI PROGRAMMAZIONE

UNIFIED MODELLING LANGUAGE

DISPENSA 15.11

15-11_OOP_UML_[12]

Versione del: 07/11/2015
Revisione numero: 12

Prof. Andrea Zoccheddu
Dipartimento di Informatica

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

UNIFIED MODELLING LANGUAGE

MODELLO U.M.L.

INTRODUZIONE

COSA È UML (UNIFIED MODELLING LANGUAGE)

UML è un acronimo anglosassone per Unified Modelling Language che significa Linguaggio di Modellazione Unificato.

In generale UML è un linguaggio «visuale» nato per progettare del software e poter rappresentare diversi modelli di programmazione; analizzando in dettaglio UML si potrebbe verificare che UML è un modo per rappresentare elementi di progettazione e le relazioni e i comportamenti degli elementi.

UML è diventato uno standard de facto per la modellazione di progetti software e la sua diffusione è accresciuta anche in ambienti non direttamente legati agli oggetti.

Definito nel 1994, nel 1997 è stato ammesso come standard dall'Object Managemt Group (OMG).

I DIAGRAMMI UML

Un diagramma è un modo per rappresentare graficamente cose e anche le relazioni tra queste cose. Le cose possono essere rappresentazioni ideali di oggetti del mondo reale, ma anche elementi astratti che costituiscono un software ma anche descrizioni del comportamento di enti.

In UML i diagrammi sono divisi in due categorie:

- diagrammi strutturali e
- diagrammi comportamentali.

I diagrammi strutturali servono per rappresentare l'organizzazione concreta delle cose all'interno di un sistema e come queste cose si relazionano le une con le altre. UML annovera tra i diagrammi di questa categoria i seguenti:

- diagrammi di classe;
- diagrammmi di oggetto;
- diagrammi di componente;
- diagrammi di struttura composta;
- diagrammi di deployment;
- diagrammi di package.

I diagrammi comportamentali servono per rappresentare il comportamento delle cose all'interno di un sistema, come requisiti, operazioni, e cambiamenti interni di stato. UML annovera tra i diagrammi di questa categoria i seguenti:

- diagrammi di attività;
- diagrammi di comunicazione;
- digrammi di interazione;
- diagrammi di sequenza;
- diagrammi di stato;
- diagrammi di temporizzazione.

COME USARE UML

In questo corso useremo UML come semplice modello di rappresentazione della programmazione ad oggetti, utile per descrivere classi e istanze che costituiscono gli elementi del progetto software. Per questo ci concentreremo sui seguenti aspetti descrittivi:

- diagrammi di classe;
- diagrammmi di oggetto;

RIEPILOGO

Diagramma è una rappresentazione grafica di elementi concreti o astratti.

UML significa Unified Modelling Language, ovvero Linguaggio di Modellazione Unificato.

UML è un modo per rappresentare graficamente, mediante diagrammi, elementi di sistemi.

UML consente di rappresentare classi ed oggetti (istanze) con diagrammi.

UML utilizza due categorie di diagrammi: strutturali e comportamentali.

I diagrammi strutturali servono per rappresentare le cose, la loro organizzazione e le relazioni.

I diagrammi comportamentali servono per rappresentare i modi di agire delle cose.

RAPPRESENTAZIONE DI UNA CLASSE

SCHEMA DI UNA CLASSE

UML usa rappresentazioni grafiche precise per descrivere oggetti e relazioni. In teoria UML consente di omettere qualsiasi elemento grafico, superando in alcuni casi le possibilità del linguaggio utilizzato per realizzare ciò che si è modellato.

Per modellare una classe si usa un rettangolo diviso orizzontalmente in tre sezioni:

Nome-Classe	← l'intestazione dello schema della classe che contiene il suo nome
§ attributo	← sezione degli attributi della classe, i dati
§ metodo	← sezione dei metodi della classe, i comportamenti

L'intestazione contiene il solo nome della classe. Per convenzione è centrato nella sezione.

La seconda sezione è l'elenco degli attributi che prevede la seguente nomenclatura:

- gli attributi iniziano sempre con una lettera minuscola
- gli attributi privati sono preceduti da un segno meno (-)
- gli attributi pubblici sono preceduti da un segno più (+)
- gli attributi protetti sono preceduti da un segno cancelletto (#)
- gli attributi interni sono preceduti da un segno tilde (~)
- dopo il nome dell'attributo si scrive il suo tipo preceduto da due punti (: tipo)

La terza sezione è l'elenco dei metodi che prevede la seguente nomenclatura:

- i Metodi iniziano sempre con una lettera Maiuscola
- i metodi privati sono preceduti da un segno meno (-)
- i metodi pubblici sono preceduti da un segno più (+)
- i metodi protetti sono preceduti da un segno cancelletto (#)
- i metodi interni sono preceduti da un segno tilde (~)
- se un metodo ha parametri essi si scrivono tra parentesi tonde dopo il nome
- se un metodo restituisce un tipo si scrive con due punti seguito dal tipo (: tipo)

ESEMPIO

Proviamo a rappresentare una classe per modellare un Mostro per un videogioco. Il Mostro per il momento è un tipo di dato astratto (una classe) ma senza alcuna informazione o comportamento.

Mostro	← il nome della classe è Mostro
	← la classe non ha attributi
	← la classe non ha Operazioni

Se invece il nostro Mostro deve essere descritto con attributi e metodi, questi si scrivono nelle opportune sezioni, preceduti dall'operatore di visibilità:

Mostro	← il nome della classe è Mostro
- nome : string - velocità : int - visibile : bool + preda : bool	← la classe ha tre attributi privati e uno pubblico
+ Insegui ( ) + Fuggi ( ) - Cura ( ) + Nascondi ( )	← la classe ha tre metodi pubblici e uno privato

Questa classe descrive un Mostro che è descritto da un nome e altri attributi e da comportamenti come, per esempio, tecniche di fuga e di inseguimento.

RAPPRESENTAZIONE DEI COSTRUTTORI

In UML i costruttori appaiono come metodi (in realtà sono tutte Operazioni per UML) ma UML distingue l'utilizzo delle due operazioni. I costruttori possono essere anch'essi privati, protetti o pubblici.

Mostro	Sezione Nome
- nome : string - velocità : int - visibile : bool + preda : bool	Sezione attributi
+ Mostro ( ) + Insegui ( ) + Fuggi ( ) - Cura ( ) + Nascondi ( )	← il costruttore si scrive tra le Operazioni

L'overload di metodi e costruttori non ha alcuna particolare simbologia UML. È sufficiente scrivere i metodi o i costruttori omonimi rispettando i vincoli del linguaggio che si intende usare.

PARAMETRI

I parametri dei metodi e dei costruttori si scrivono dentro le parentesi; ogni parametro deve anche specificare la «direzione» del parametro con le parole in oppure out oppure inout.

La parola in significa che il parametro richiede un dato ma non modifica l'argomento; la parola out significa che il parametro non richiede un dato ma modifica la locazione dell'argomento; la parola inout significa che il parametro richiede un dato e modifica l'argomento (come ref).

Per esempio il metodo Fuggi del Mostro potrebbe essere scritto:

Mostro	Sezione Nome
- nome : string - velocità : int - visibile : bool + preda : bool	Sezione attributi
+ Mostro (in nome : string, out visibile : bool) + Insegui ( ) + Fuggi (in secondi : int) - Cura ( ) + Nascondi ( )	Sezione Operazioni

TIPO RESTITUITO

Il tipo restituito da un metodo si scrive dopo le parentesi tonde, preceduto da due punti:

Mostro	Sezione Nome
- nome : string - velocità : int - visibile : bool + preda : bool	Sezione attributi
+ Mostro (in nome : string, out visibile : bool) + Insegui ( ) + Fuggi (in secondi : int) + Velocità ( ) : int + Fermo () : bool	Sezione Operazioni

CARATTERISTICHE STATICHE

Secondo la nomenclatura UML, gli elementi statici sono contraddistinti da una sottolineatura, sia per attributi che per operazioni.

Mostro	Sezione Nome
- nome : string - velocità : int - visibile : bool - creati : int	Sezione attributi
+ Mostro (in nome : string, out visibile : bool) + Mostro ( ) + Insegui ( ) + Fuggi (in secondi : int) + Creati () : int	Sezione Operazioni

CLASSI STATICHE

Non ho ancora scoperto come rappresentare le classi statiche, ma mi pare una buona idea sottolineare il nome della classe stessa.

RAPPRESENTAZIONE DI PROPRIETÀ

Nel linguaggio UML non è prevista una nomenclatura per le proprietà.

In questa dispensa si propone di utilizzare il concetto di Stereotipo di UML che consente di modificare il significato di un elemento di un diagramma. Uno stereotipo si rappresenta racchiudendo tra parentesi acute (angolari) il concetto che si desidera applicare all'elemento. Quindi per una proprietà è consentito scrivere: <Proprietà> attributo.

Mostro	Sezione Nome
- nome : string <Proprietà> + Velocità : int <Proprietà> + Visibile : bool	Sezione attributi
+ Mostro (in nome : string) + Insegui ( ) + Fuggi (in secondi : int)	Sezione Operazioni

In questo esempio si omettono i metodi di accesso (set e get) da associare alla proprietà. È anche possibile usare una proprietà come metodo in lettura per ottenere informazioni senza che l'utilizzatore si renda conto che sia un metodo (sembra un attributo). Quindi potremmo scrivere:

Mostro	Sezione Nome
- nome : string - velocità : int - visibile : bool	Sezione attributi
+ Mostro (in nome : string) + Insegui ( ) + Fuggi (in secondi : int) <Proprietà> + Velocità (): int <Proprietà> + Visibile () : bool	Sezione Operazioni

Si osservi che, a differenza degli attributi, una proprietà si scrive con la Maiuscola iniziale.

RELAZIONI TRA CLASSI

UML prevede cinque diversi tipi di relazione tra le classi:

- Dipendenza (la più debole delle relazioni)
- Associazione
- Aggregazione
- Composizione
- Generalizzazione (la più forte delle relazioni)

DIPENDENZA (ALFA USA UN BETA)

La relazione di dipendenza è quella più debole tra i 5 tipi di relazione ammessi da UML.

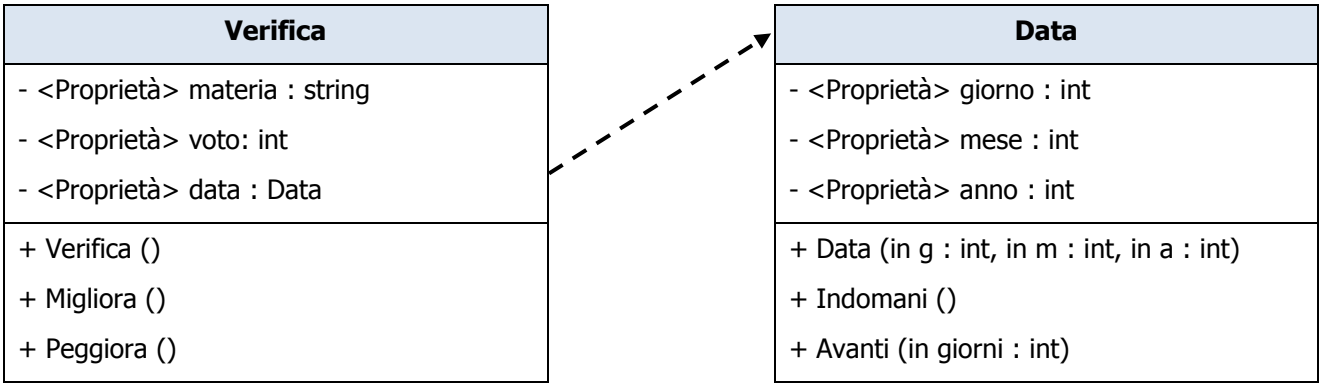
La dipendenza serve per indicare che una classe utilizza, oppure ha conoscenza di, un'altra classe; tipicamente, la relazione si legge come "… **usa un** …".

Questa relazione è per definizione di tipo transitorio intendendo con questo che la classe "dipendente" interagisce brevemente con la classe da cui dipende, e non mantiene con essa nessuna relazione per un lungo periodo di tempo.

Un caso di dipendenza si ha quando una classe utilizza un dato descritto come un'altra classe. Per esempio una classe Verifica, che modella eventi come interrogazioni o compiti in classe, potrebbe usare un dato di tipo Data, a sua volta implementato come un oggetto.

La dipendenza è una relazione a livello di modello e non a livello di esecuzione per cui, in sostanza, essa descrive la necessità di tenere conto di possibili cambiamenti del "fornitore" (la classe usata), le cui eventuali variazioni possono ripercuotersi sul "cliente" (la classe utilizzatrice).

Nel diagramma UML è rappresentata come una linea tratteggiata con una freccia che va dall'elemento dipendente verso quello usato.



L'attributo (proprietà) data della classe Verifica usa la classe Data per implementare la data. Quindi esiste una dipendenza debole tra Verifica (cliente) e Data (fornitore).

ASSOCIAZIONE (ALFA HA UN BETA)

La relazione di associazione è appena più forte della precedente.

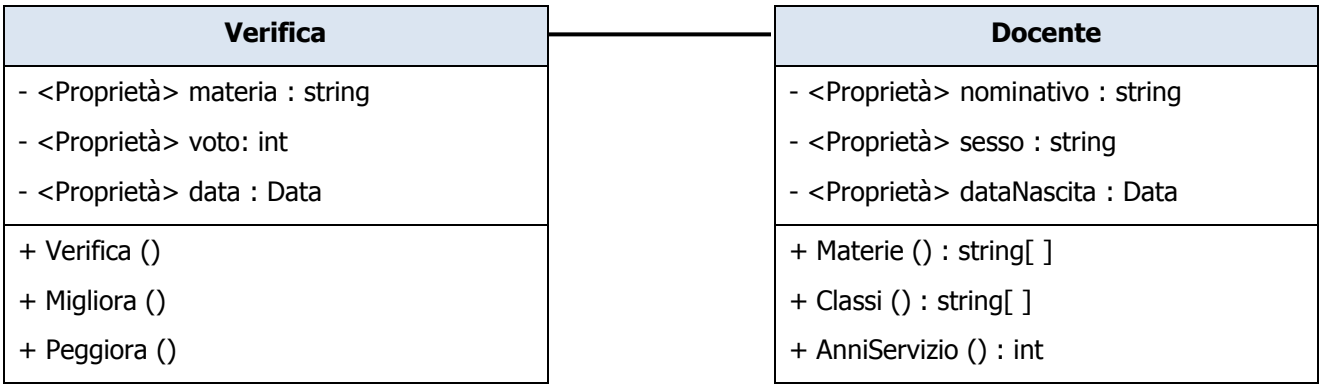
L'associazione serve per rappresentare la condizione in cui una classe ha una relazione con un'altra classe e la mantiene per un consistente periodo di tempo.

In UML l'associazione è rappresentata con una linea continua che collega le due classi.

Si può concettualizzare una associazione come un evento del tipo "… ha un …" e la si può pensare come un riferimento stabile (puntatore) ad un altro oggetto in cui si ha la possibilità di definire anche la direzione del riferimento, che può essere unidirezionale o bidirezionale.

Non è detto che il ciclo di vita dei due oggetti sia parallelo per cui uno dei due può essere distrutto mentre l'altro prosegue la sua esistenza.

Per esempio possiamo ipotizzare che una verifica corrisponda ad un Docente, per cui si ha:



ASSOCIAZIONE NAVIGABILE

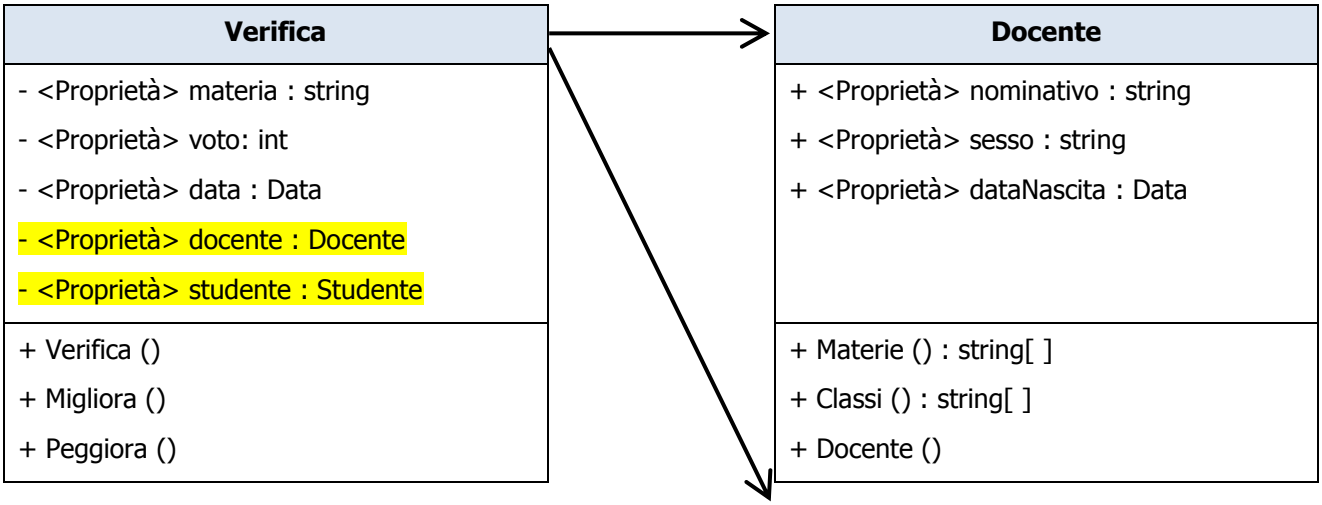
La navigabilità serve per rappresentare la condizione in cui è possibile transitare da una classe verso un'altra classe e lo consente per un consistente periodo di tempo. In effetti la navigabilità è un rafforzamento della associazione, che diventa «navigabile».

In UML la rappresentazione della navigabilità si riproduce con una freccia a una o entrambe le estremità.

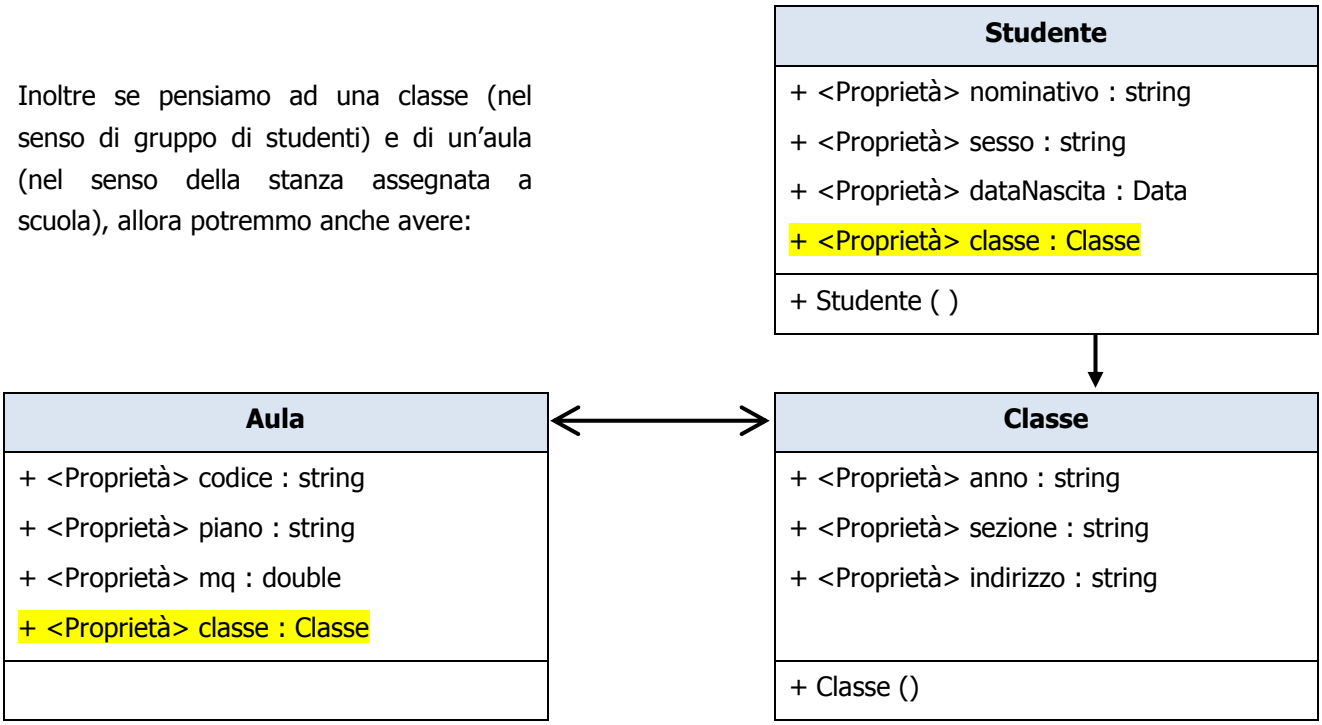
L'associazione diviene navigabile aggiungendo un attributo di tipo corrispondente alla classe associata. Aggiungere un attributo ad una classe con un riferimento all'altra significa rendere "Navigabile"

l'associazione, ovvero esiste una notazione esplicita che indica la possibilità di "navigare" da una categoria all'altra mediante l'attributo.

Una freccia in una sola direzione implica che è possibile navigare solo da una classe verso l'altra, ma non viceversa. Se è possibile navigare in entrambe le direzioni, allora si utilizza una doppia freccia. Se possiamo ipotizzare che da una verifica si possa navigare verso un Docente ma anche verso uno studente, allora si ha:

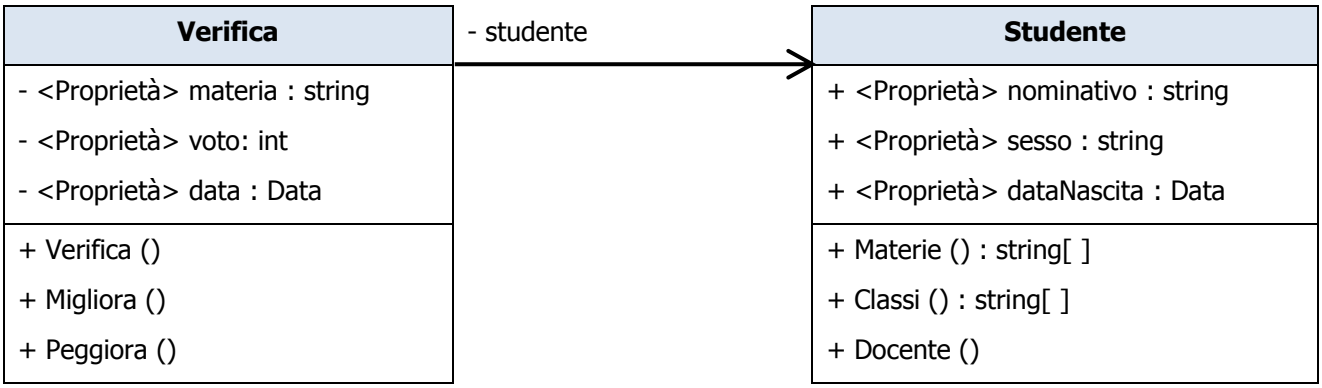


Inoltre se pensiamo ad una classe (nel senso di gruppo di studenti) e di un'aula (nel senso della stanza assegnata a scuola), allora potremmo anche avere:



Per indicare i nomi degli attributi che rappresenteranno l'associazione, questi invece di essere normalmente scritti all'interno, si scrivono a fianco del rettangolo, vicino alla freccia.

Quindi un'associazione potrebbe diventare come segue:



MOLTEPLICITÀ DI UNA ASSOCIAZIONE

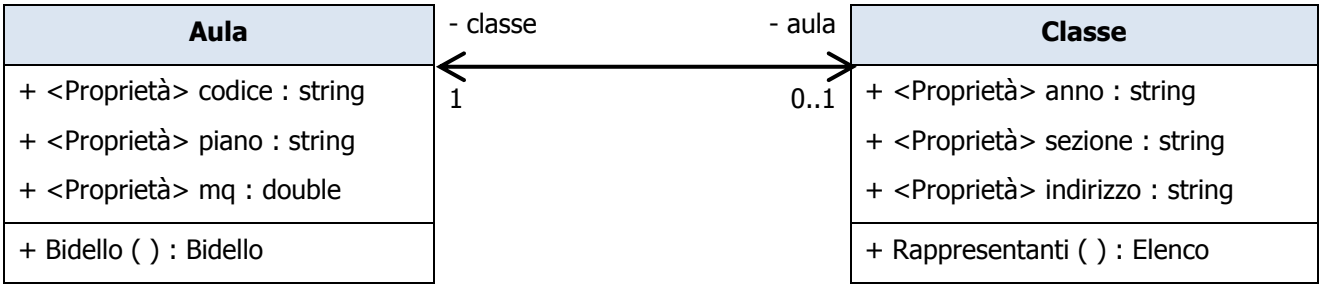
La molteplicità di un'associazione risponde alla seguente domanda: **presa un'istanza di una classe, quante istanze dell'altra classe sono correlate tramite l'associazione?**

La risposta a questa domanda può assumere uno dei valori mostrati nella seguente tabella:

Notazione	Significato	Nome
1 o niente	solo e sempre una (di default)	Molteplicità Singola
0..1	minimo 0 e massimo 1	Molteplicità Singola
M..N	minimo M e massimo N	Molteplicità
0 * o solo *	da 0 in su	Molteplicità
1..*	da 1 in su	Molteplicità

La molteplicità è riportata a fianco delle classi sotto la linea dell'associazione; se non si specificano dei valori di molteplicità, si assume che essa sia 1 per entrambi i lati.

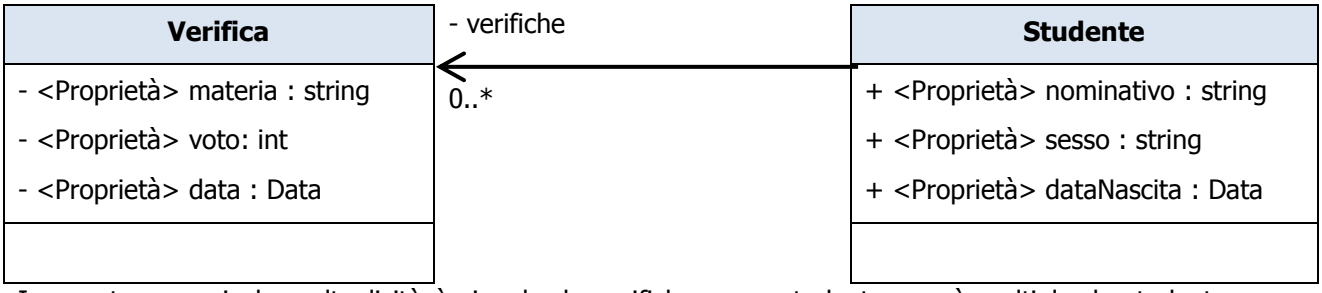
MOLTEPLICITÀ SINGOLA



In questo esempio la molteplicità è singola in entrambi i versi, anche se data una classe le corrisponde sempre una e una sola aula, mentre viceversa data un'aula può corrispondere una classe ma anche nessuna.

MOLTEPLICITÀ MULTIPLA

In alcuni casi, considerata un'istanza di una classe è possibile che le corrispondano più istanze di un'altra classe, mediante l'associazione; è il caso delle verifiche di uno studente (dato uno studente gli sono associate più verifiche, da nessuna a molte) anche se data una verifica è associata solo ad uno studente (singola).



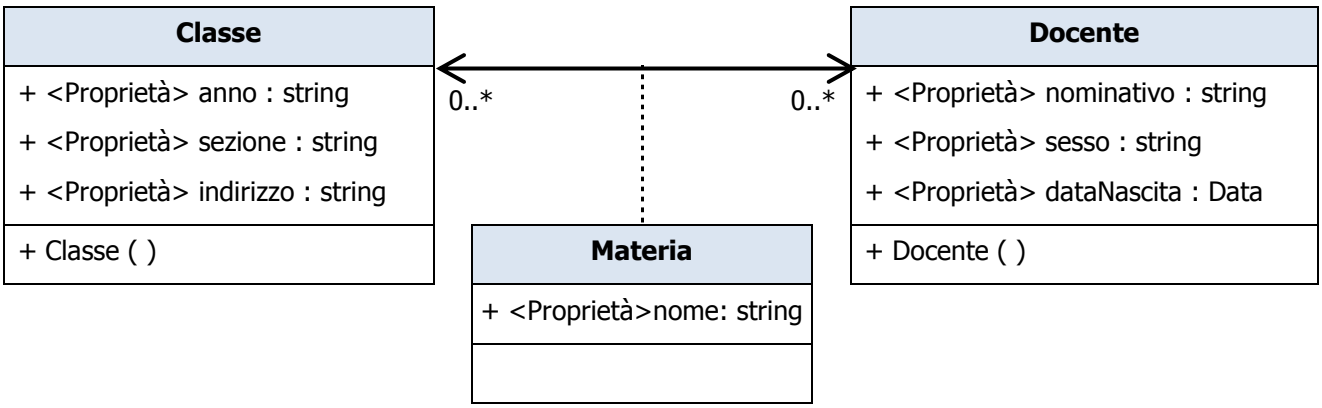
In questo esempio la molteplicità è singola da verifiche verso studente, ma è multipla da studente verso verifiche, anche se allo studente potrebbero corrispondere zero verifiche.

MOLTEPLICITÀ MULTIPLA CON CLASSE

Talvolta, la associazione che correla gli elementi delle due classi non è un semplice collegamento, ma rappresenta una situazione in cui c'è la necessità di esprimere maggiori informazioni a riguardo. In UML si può definire un'ulteriore classe che descrive i dati e i comportamenti aggiuntivi della relazione. In questo caso si introduce una classe di associazione che viene collegata all'associazione principale.

La classe di associazione è rappresentata come una normale classe, con un rettangolo e le tre sezioni, ma è connessa alla linea di associazione invece che a una classe e la riga di connessione è tratteggiata.

Per esempio se pensiamo che un docente possa insegnare in più classi e in una classe insegnano più docenti, la associazione è del tipo (0..*)- (0..*) a cui però occorre aggiungere l'informazione di quale materia insegna.



La classe Materia in questo esempio viene introdotta per modellare esclusivamente l'associazione tra Docente e Classe. Se la classe Materia avesse bisogno di esistere per altri motivi (per es. per modellare altre situazioni come le verifiche) allora forse la classe di associazione dovrebbe essere qualcosa di diverso. Per esempio si potrebbe immaginare di non legare direttamente Docente e Classe, ma introdurre una nuova classe Insegnamento che sia legata a Docente e poi, separatamente, a Classe.

AGGREGAZIONE

La relazione di aggregazione è ancora più forte della precedente.

Si tratta di situazioni dove le associazioni tra classi stanno ad indicare che gli oggetti di una classe sono composti o costituiti da oggetti di un'altra classe.

In senso stretto, le notazioni di aggregazione e di composizione non sono un diverso tipo di relazione, ma evidenziano piuttosto la natura della stessa che viene messa in evidenza ed espressa chiaramente all'interno dell'associazione.

Un'aggregazione è utilizzata per indicare che, oltre ad avere i consueti propri attributi, una classe può includere come caratteristiche istanze di altre classi; la relazione si legge tipicamente come "... possiede un ...".

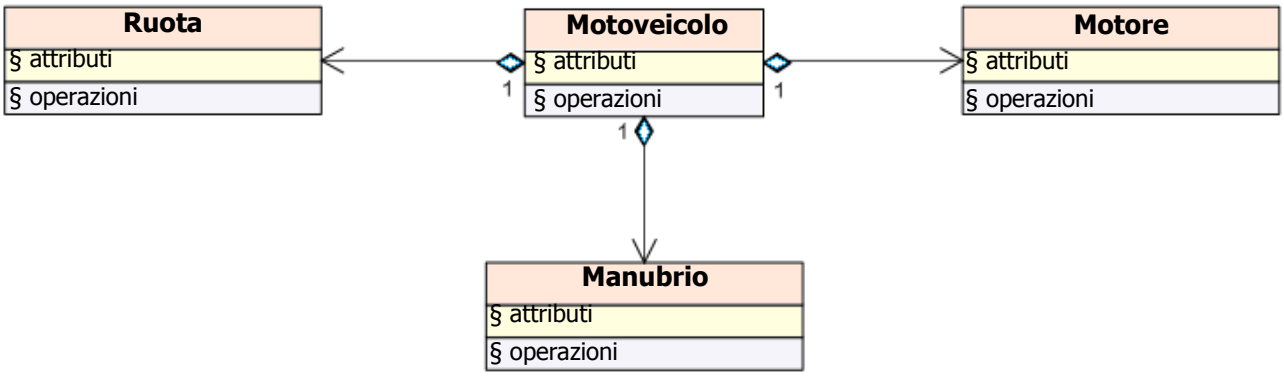
L'aggregazione va a definire una relazione del tipo "tutto-parti", nel senso che un oggetto, che nella relazione rappresenta il "tutto", è composto da più parti, costituite da oggetti di altri tipi. Secondo questa concezione non è possibile che un'aggregazione sia ricorsiva (una classe composta da sé stessa come sua parte) nemmeno attraverso cicli (una prima classe composta da una seconda che a sua volta contiene la prima).

L'aggregazione è una situazione più forte dell'associazione e, al contrario di questa, implica che il possesso può durare anche per l'intero ciclo di vita dell'oggetto, ma anche che la distruzione del "tutto" non comporta necessariamente la distruzione delle "parti".

A questo punto è ovvio che la navigabilità e la molteplicità risiedono solo in un lato del diagramma.

Nell'aggregazione un componente costitutivo del "tutto" può esistere anche prima di essere parte di un composto per cui è possibile assemblare il tutto a partire da componenti che preesistevano prima della sua costituzione.

In UML un'aggregazione viene rappresentata con una linea continua con un rombo vuoto agganciato all'estremità dalla parte della classe che rappresenta il "tutto".



Ovviamente un "tutto" può essere formato da più "parti" (come una bicicletta da più ruote), ma una "parte" può contribuire a formare un solo "tutto" (una ruota può partecipare per formare al più una bicicletta, e non anche altre).

Per implementare in Visual C# una aggregazione si segue generalmente la stessa modalità di una ordinaria associazione, sebbene l'attributo o la proprietà sia sempre di tipo classe. L'attributo quindi è un tipo riferimento che collega l'ente principale all'oggetto che la compone.

COMPOSIZIONE

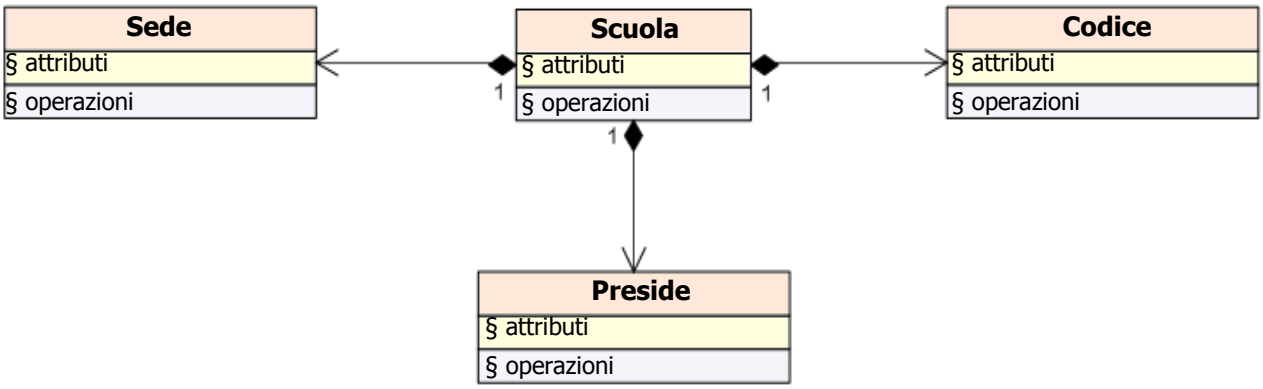
La composizione, in modo analogo all'aggregazione, rappresenta una relazione del tipo"tutto-parti", ma le istanze dei componenti sono indissolubilmente legate, a differenza della prima. Quindi composto e componenti hanno lo stesso ciclo di vita. Quando viene creato l'organismo (il "tutto") si devono contestualmente creare anche i suoi componenti (le "parti") e quando l'organismo (il "tutto") viene distrutto, anche i suoi componenti (le "parti") sono distrutti.

Questa situazione comporta che la composizione ha sempre una molteplicità singola, di 1 oppure 0..1, poiché al massimo un oggetto può essere responsabile del ciclo di vita dell'altro oggetto.

Nell'aggregazione questo non vale ed un componente può esistere senza essere parte di un assemblaggio così come l'assemblaggio può essere costituito da componenti che esistevano prima della sua costituzione. Quindi la composizione rappresenta una forma più rigorosa dell'aggregazione.

Una relazione di composizione si legge come " B costituisce A", il che significa che il "tutto" (A) è composto (formato) da "parti" (B).

In UML la notazione per la composizione è simile a quella dell'aggregazione ma il rombo è pieno invece che vuoto.



Per implementare in Visual C# una composizione si segue ancora la stessa modalità di una composizione, sebbene i metodi costruttori e distruttori debbano preoccuparsi di gestire i cicli di vita delle parti.

GENERALIZZAZIONE

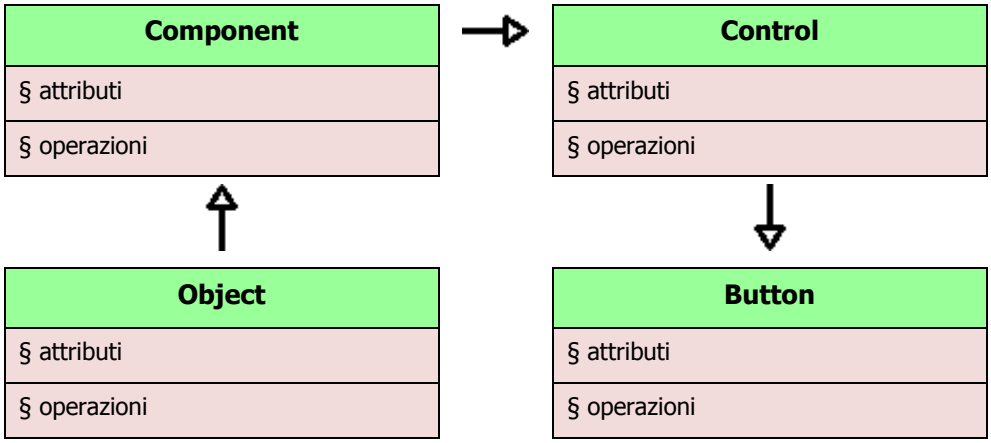
La generalizzazione va a descrivere un tipo diverso di relazione tra classi. Non c'è più una relazione tra elementi di categorie diverse, ma elementi che appartengono alla stessa categoria anche se sono classi diverse.

Una classe (detta sottotipo) è un caso particolare di una altra classe (detta supertipo).

La generalizzazione UML va a rappresentare la gerarchia tra classi derivate tipica della OOP. Questo significa che una classe derivata è un caso particolare di classe base.

In UML è rappresentata con una linea continua, con una freccia dalla punta larga e vuota che si dirige dalla classe specializzata verso le classi più generali. La relazione può essere letta dalla specializzazione verso la generalizzazione nel modo "… è un tipo di …".

Un esempio quindi può essere quello di Studenti che sono anche Persone, sebbene non tutte le persone siano Studenti. Il caso più emblematico di gerarchia si ritrova nella programmazione ad oggetti dei componenti di .NET. Per esempio tutti gli elementi delle librerie sono derivate dalla classe Object; la classe Control deriva dalla classe Component; la classe Button deriva dalla classe Control.



RIEPILOGO

In **UML**, tra le categorie sono possibili **cinque** diverse relazioni riassunte nella tabella seguente:

Relazione	Intensità	Concetto	Tempo	Segno
Dipendenza	Debole	un B è usato da un A	Periodo breve	
Associazione		un B è tenuto da un A	Periodo lungo	
Aggregazione		un B appartiene a un A	Intera vita, ma risolubile	
Composizione	Forte	un B costituisce un A	Intera vita indissolubilmente	
Generalizzazione	Gerarchia	un B è un A	Condivide l'esistenza	

La tabella riassume tutti i tipi di relazione contemplate da UML e descrive le rispettive intensità, significati, durata e simbolo utilizzato.

È opportuno ricordare che UML non nasce come modello di rappresentazione della OOP ma che si presta comunque per costruire schemi di rappresentazione di software tra cui programmazione OOP così come progetti concettuali di basi di dati.

L'associazione ha una ulteriore classificazione in base alla molteplicità tra le classi:

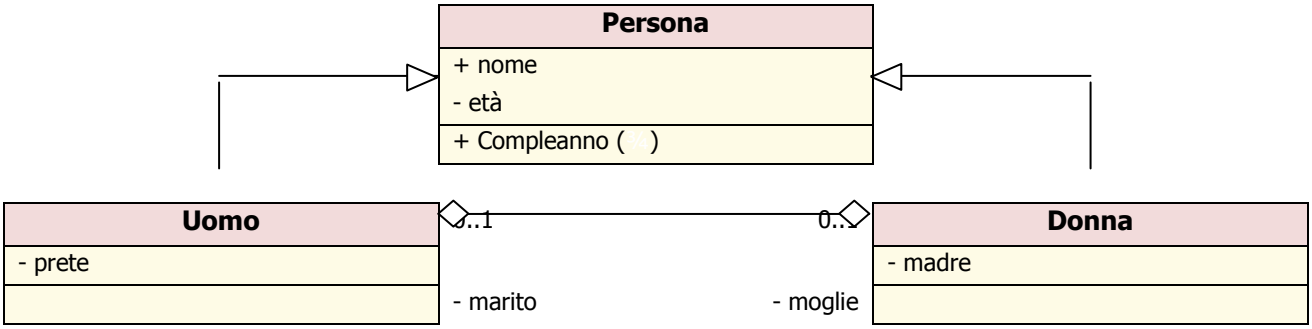
1 (o niente)	0..1	M..N	0..* (oppure *)	1..*

Le molteplicità sono utilizzate anche nelle associazioni di maggiore intensità, salvo la generalizzazione

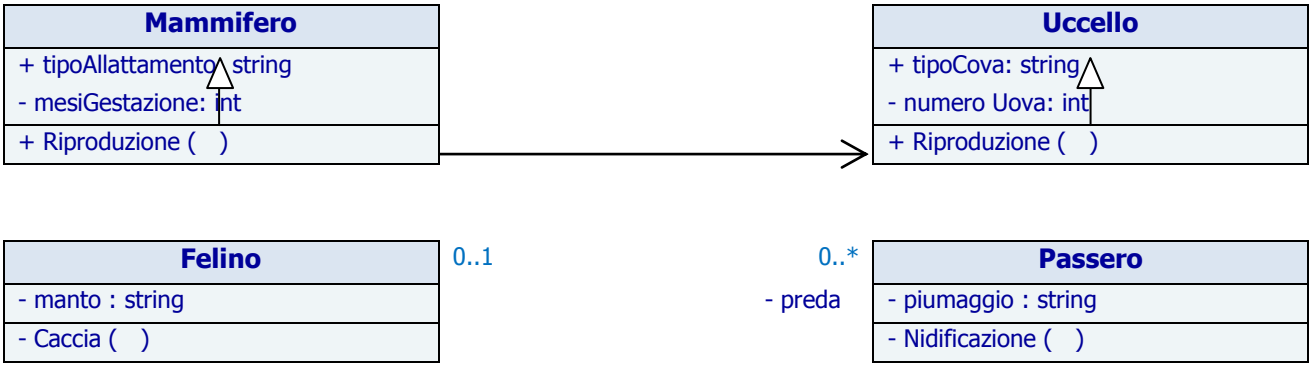
ESEMPI

ESEMPI

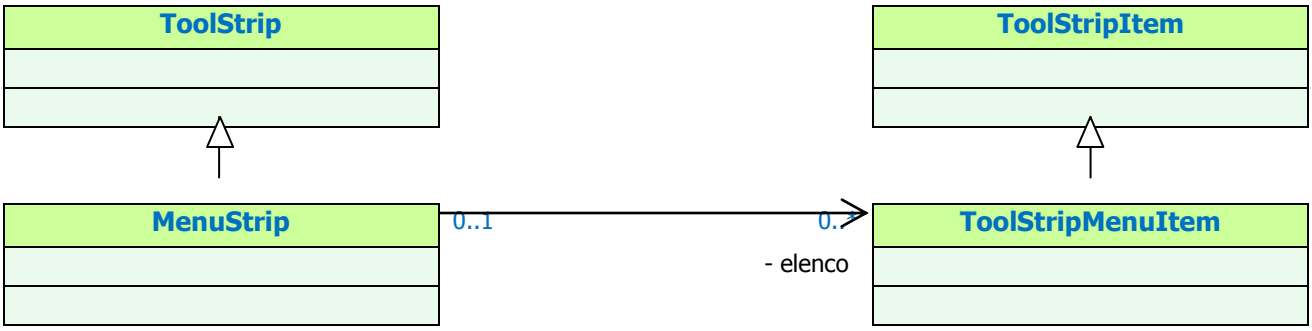
ESEMPIO 1. PERSONE



ESEMPIO 2. ANIMALI



ESEMPIO 3. MENU



RICHIAMI DI GERARCHIA

DEFINIZIONI

Prima di concludere questa parte di discussione, è preferibile ricordare le definizioni che si possono riferire alle classi di una gerarchia.

Superclasse	Sottoclasse
Supertipo	Sottotipo
Classe Base	Classe Derivata
Classe Genitore	Classe Figlio
Antenato	Discendente
Per Antenato si intende una qualsiasi classe superiore nella Gerarchia (un avo)	Per Discendente si intende una qualsiasi classe inferiore nella Gerarchia (un postero)

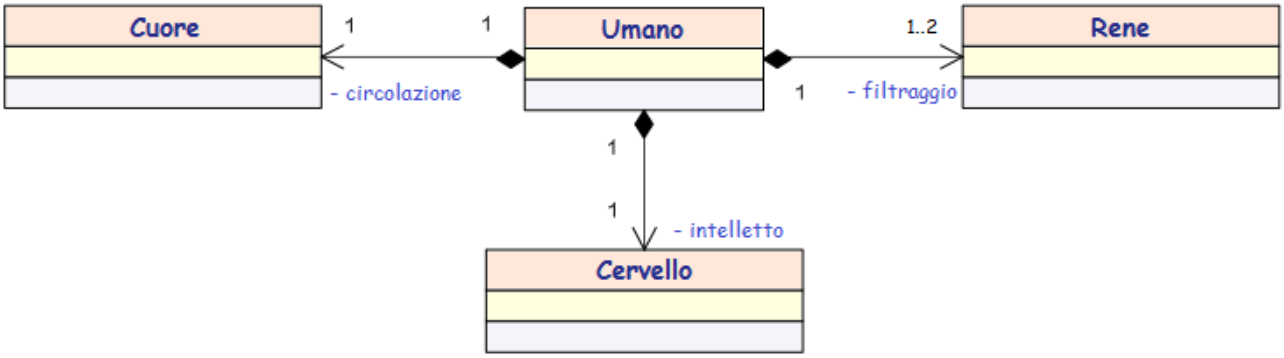
È opportuno inoltre osservare che, in alcuni linguaggi, è possibile che una classe sia derivata da più di un antenato; poiché in Visual C# questo non è possibile (o almeno senza utilizzare altri strumenti come le Interfacce) non approfondiremo questa questione.

ESERCIZI

ESERCIZI

USARE LE RELAZIONI

Esercizio 1) RICONOSCERE LE RELAZIONI



Esercizio 2) NATANTE

- Rappresentare in UML le classi **Barca**, **Scalmo** e **Remo** e rappresentare le relazioni opportune che legano tra loro gli elementi delle classi.

Esercizio 3) INTERROGAZIONI 1

- Rappresentare in UML le seguenti classi:
  1. Studente con attributi nominativo, età e sesso;
  2. Docente con attributi nominativo e anzianità di carriera;
  3. Verifica che indica una interrogazione orale con data, voto e materia
- Rappresentare le opportune relazioni che legano tra loro gli elementi delle classi.

Esercizio 4) INTERROGAZIONI 2

- Rappresentare in UML le seguenti classi:
  1. Studente con attributi nominativo, età e sesso;
  2. Docente con attributi nominativo e anzianità di carriera;
  3. Materia che indica la materia oggetto di un'interrogazione orale
  4. Verifica che indica una interrogazione orale con data e voto
- Rappresentare le opportune relazioni che legano tra loro gli elementi delle classi.

Esercizio 5) CAMPIONATO A

- Rappresentare in UML le seguenti classi:
  1. Squadra con attributi nome, città, colori;
  2. Partita con attributi data;
  3. La relazione che intercorre tra la squadra che gioca in casa e una partita
  4. La relazione che intercorre tra la squadra che gioca ospite e una partita
- Rappresentare le opportune relazioni che legano tra loro gli elementi delle classi.

Esercizio 6) CAMPIONATO B

- Rappresentare in UML le seguenti classi:
  1. Squadra con attributi nome, città, colori;
  2. Partita con attributi data, gol delle due squadre;
  3. Il metodo GolCasa che incrementa di +1 i gol della Squadra di casa;
  4. Il metodo GolOspite che incrementa di +1 i gol della Squadra ospite;
  5. I metodi Vittoria, Pareggio e Sconfitta.
- Rappresentare le opportune relazioni che legano tra loro gli elementi delle classi e in particolare le relazioni che intercorrono tra la squadra che gioca ospite e una partita.

Esercizio 7) CAMPIONATO C

- Rappresentare in UML le seguenti classi:
  1. Squadra con attributi nome, città, colori;
  2. Calciatore con attributi nominativo, maglia e gol
  3. Portiere derivata da Calciatore, con attributo parate;
  4. Attaccante derivata da Calciatore, con attributo assist;
- Rappresentare le opportune relazioni che legano tra loro gli elementi delle classi.

Esercizio 8) BASKET

- Rappresentare in UML le seguenti classi:
  1. Squadra con attributi nome, città, colori;
  2. Cestista con attributi nominativo, numero e i tiri1, tiri2 e tiri3;
  3. Playmaker derivata da Cestista, con attributo passaggi;
  4. Pivot derivata da Cestista, con attributo rimbalzi;
- Rappresentare le opportune relazioni che legano tra loro gli elementi delle classi.

Esercizio 9) SCACCHI

- Rappresentare in UML le seguenti classi:
  1. Giocatore con attributi nome, città, soprannome;
  2. Partita con attributi data e orari di inizio e fine;
  3. La relazione di chi usa i bianchi e i neri;
  4. Il metodo Vittoria ( )

Esercizio 10) ARMA

- Rappresentare in UML le seguenti classi:
  1. Militare con attributi nome, corpo, specializzazione;
  2. Soldato (caso particolare di Militare);
  3. Sergente (caso particolare di Militare) con attributo "specialità";
  4. Tenente (caso particolare di Militare) con attributo "corpo";
  5. Drappello (gruppo di militari) con attributi nome e codice, formato da 8 soldati, comandato da un Tenente con l'aiuto di un Sergente;
  6. Missione (attività realizzata dai drappelli) con attributi data e luogo, realizzata anche da più Drappelli; un drappello può impegnarsi in più di una Missione, ma anche nessuna;

SOMMARIO